Lab 6: VGA Signals and Color Raster Picture

EEL 4712 – Spring 2012

Objective:

The objective of this lab is to study the generation of the control signals required by a VGA monitor by creating a component VGA_sync_gen that generates those signals. Using VGA_sync_gen and additional components, a simple color raster image will be displayed on a VGA monitor.

Required tools and parts:

QuartusII software package, ModelSim, UF-4712 board, and VGA Monitor.

Device(s) Used: Altera Cyclone II EP2C8T144C8 FPGA, Quartus altsyncram component, brom.mif.

An altsyncram component will be used in this lab, which requires memory that is available on the the Cyclone II EP2C8T144C8 FPGA. All output bits of this lab should be directed to the appropriate pins of the UF-4712 VGA connector. Also, a memory initialization file (brom.mif) will be used to initialize the memory values of the ROM.

References: UF-4712 Board Manual

Introduction:

The VGA monitor connector on the UF-4712 board consists of five signals, RED, GREEN, BLUE, HORIZ_SYNC, and VERT_SYNC. Look into the UF-4712 Board Guide for the header pin outs for the UF-4712 board for these 5 signals. The timing relationships among this signal are shown in **Figures 1 and 2** at the end of this lab write-up. The generation of signals needed for the raster on the VGA monitor begins with dividing the frequency of the 25.175 MHz clock on the UF-4712 board down to the horizontal and vertical sync frequencies. This means your design will contain a ***horizontal counter*** and a ***vertical counter***. The horizontal and vertical sync pulses of appropriate lengths are then produced from the two counters.

Horizontal and vertical synchronization. A video display consists of 640 pixels in the horizontal direction and 480 lines of pixels in the vertical direction. The monitor starts each refresh cycle by updating the pixel in the top left-hand corner of the screen, which can be treated as the origin (0,0) of an X–Y plane. After the first pixel is refreshed, the monitor refreshes the remaining pixels in the row. When the monitor receives a pulse on the ***HORIZ_SYNC*** pin, it refreshes the next row of pixels. The time required for the sweep (i.e. the horizontal sweep period) is 31.77 μs. This process is repeated until the monitor reaches the bottom of the screen. When the monitor reaches the bottom of the screen, a 64 μs pulse is applied to the ***VERT_SYNC*** pin, causing the monitor to begin refreshing pixels at the top of the screen (i.e., at [0,0]). As shown in Figure 2, the VERT_SYNC pulse must be repeated every 16.6 ms (vertical sweep period). A complete screen of information is traced by the electron beam every 16.6 ms for a frame rate of 60 Hz.

Blanking intervals. In order to accommodate the time required to move the electron beam back to the left side of the screen (horizontal retrace period) the electron beam must be turned off during the retrace time or the return trace would be visible. This is accomplished by two blanking intervals during the horizontal refresh cycle marked by B and C at the beginning of the trace and E at the end of the trace in Figure 1. Similar blanking intervals are required during vertical refresh (P, Q, and S in Figure 2). The color beams are turned on only when the horizontal counter is in the period D and the vertical counter is in period R.

Hint: Blanking intervals can be easily implemented by creating a video “gate” signal (Video_On) that is true only when the color beams can be active. Refer to the figure 3 at the end of this lab write-up for a graphical look at the VGA display signals.

Pre-lab requirements:

1. VGA Sync Component (VGA_sync_gen)

Create a VHDL entity called VGA_sync_gen.

- Be sure to include the IEEE.NUMERIC_STD package.
- Be sure to include the work.VGA_LIB package, which is a custom package for this project.
- Some useful constants are provided in Figure 4, and are included in the provided VGA_LIB package.
- The entity has a clock input, a reset input, and 5 outputs: h_count, v_count, h_sync, v_sync, and video_on. It is up to you to determine the widths of each signal, but there is a hint in vga_lib.vhd.
- Create two counters (it's up to you to determine the width).
  - **Hcount**
    - Counts up to the horizontal period (H_MAX – See Figure 4) using the 25.175 MHz on-board oscillator.
    - IMPORTANT- A value of zero on Hcount corresponds to the beginning of section D in Figure 1.
  - **Vcount**
    - Counts up to the vertical period (V_MAX – See Figure 4). It will increment at a particular point in the horizontal counters count (Hcount = H_VERT_INC, See Figure 4).
    - IMPORTANT- A value of zero on Vcount corresponds to the beginning of section R in Figure 2.
- The values of video_on, h_sync and v_sync are determined by decoding the values of Hcount and Vcount and comparing the counter values to the constants provided in Figure 4.

Create a testbench that illustrates the correct behavior and timing of all outputs. First focus your simulations on the horizontal refresh cycle, since it can be simulated in its entirety easily. Then try to measure the length of VERT_SYNC (64 μs) in simulation, and do a full simulation of the vertical refresh cycle.

Turn in all vhdl files and annotated simulation results showing the correct behavior of each VGA_sync_gen output.

2. Display of Raster Picture

In this part of lab, you will partition the screen into 8x8 pixel blocks (i.e., each block is 8x8 pixels), each of which displays a color made from a combination or Red, Green, and Blue. There will be a total of 256 blocks arranged as a 16x16 grid, which forms a 128x128 pixel image (16*8 = 128). VGA resolution is 640x480, so the pixels not used by the image must be turned off. Your circuit should allow the image to be placed in five different locations: centered (when no buttons are pressed), the top left corner (when the top left button is pressed), the top right corner (when the top right button is pressed), etc.

We will need to utilize ***three additional entities*** to implement this color raster picture: a block ROM which contains the R,G,B values for each 8x8 block, an entity that generates the current block row, and an entity which generates the current block column. They are described as follows:

a) **Block row address logic**: This takes the Vcount signal generated in Prelab step 1 and generates a 4-bit row address which identifies one of the 16 rows of the 16x16 grid. The block row address will be used as a part of the address input to the ROM discussed below. Note that this logic should take into account the position of the image based on which buttons are pressed. I recommend using an enable output that forces the color outputs to 0 during an inappropriate row.

b) **Block column address logic**: This takes the Hcount signal generated in Prelab step 1 and generates a 4-bit column address which identifies one of the 16 columns of the 16x16 grid. The block column address will be used as a part of the address input to the ROM discussed below. Note that this logic should take into account the position of the image based on which buttons are pressed. I recommend using an enable output that forces the color outputs to 0 during an inappropriate column.

c) **ROM**: The ROM (ROM) is used to contain the R,G,B values for the 16x16 grid. Thus, the size of ROM is 256x3, having an 8-bit address and a 3-bit output. It uses the row (4 bits) and column block (4-bits) addresses to form an 8-bit address and outputs the Red, Green, and Blue colors (3 bits) for that pixel.

The ROM component is made from the altsyncram component provided by Quartus II. Use the Megawizard Plugin Wizard to create a 1-Port ROM (in the memory compiler section). Use the following settings:

- Specify that the output should be VHDL
- Store the file in rom.vhd
- 'q' output bus should be 3 bits
- There should be 256 3-bit words of memory
- 'q' output port should **not** be registered
- Use defaults for everything else.
- For the memory content data file, use the provided brom.mif
- Leave everything else as default and click "Finish".

A memory initialization file (brom.mif) will contain the actual memory values of the ROM and can be edited by using the Memory Editor in Quartus (or using a text editor). Brom.mif can be downloaded from the class website and can be edited to modify the 'picture' being displayed on the VGA monitor. Using the original brom.mif file, you should see the following:

- In the top-left corner of the screen there is a white box enclosing a colored area. The colors, beginning with the first row and first column, follow the sequence: black, dark blue, green, light blue, red, magenta, yellow, and white. There are a total of 14 color blocks per row, and 14 rows of color blocks.

Tasks for Prelab 2: Design and test each part using your own testbenches (simulate each component individually) with limited simulation runs.

Turn in the design files and annotated simulation results for the above parts.

3. VGA

Create a top_level VGA entity that connects together your VGA_sync_gen, ROM, column address logic, and row address logic to make a circuit that will display a raster picture as previously described. Add any additional components or logic that may necessary (hint: enables to turn off pixels outside of the image). Create a testbench and simulate the entity to prove that your design works.

Turn in the design files and annotated simulation results for the final circuit.

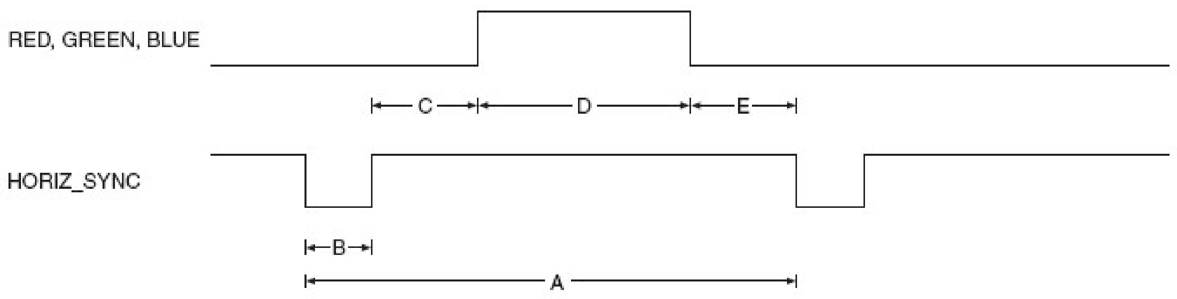
In-lab procedure:

1. Program your board with the final test setup from Prelab step 3 and connect a monitor to your UF-4712 board's VGA connector. Confirm that a picture is displayed on your VGA monitor and show your TA the functionality of the buttons.
2. Modify the brom.mif file to show a picture of your choosing. You can be creative! *Extra credit will be given to students who use complex images.*

3. **Be prepared to answer simple questions or to make simple extensions that your TA may request. If you have done the pre-lab exercises, these questions should not be difficult.**

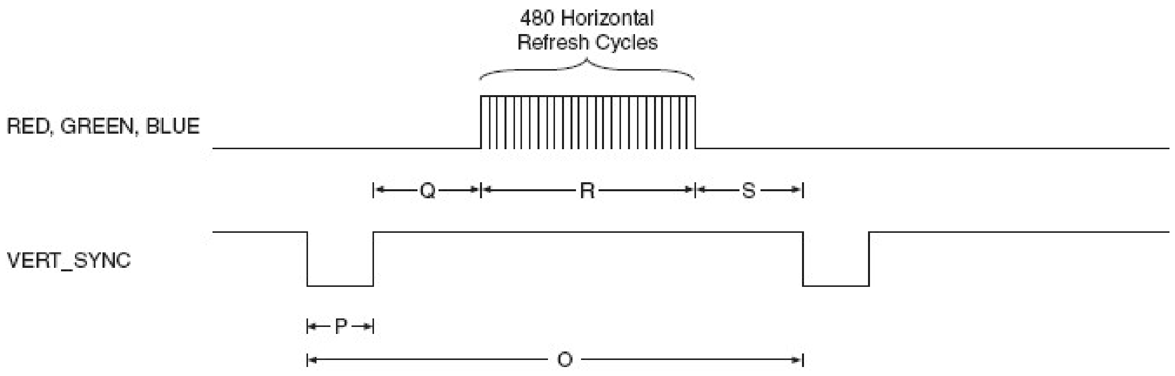
In-lab Option:

In order to test your VGA_sync_gen component you may want to design a simple component, Colorbars, which displays three vertically-aligned Red, Green, and Blue bars across the screen. This component takes Hcount as an input, and gives Red, Green, and Blue signals as outputs. It can be easily made using an IF statement. This component is not required, but will allow students to get partial credit if they cannot get their full design working.



Parameters	A	B	C	D	E
Time	31.77 µs	3.77 µs	1.89 µs	25.17 µs	0.94 µs

Figure 1. Horizontal Refresh Cycle.



Parameters	O	P	Q	R	S
Time	16.6 ms	64 µs	1.02 ms	15.25 ms	0.35 ms

Figure 2. Vertical Refresh Cycle.

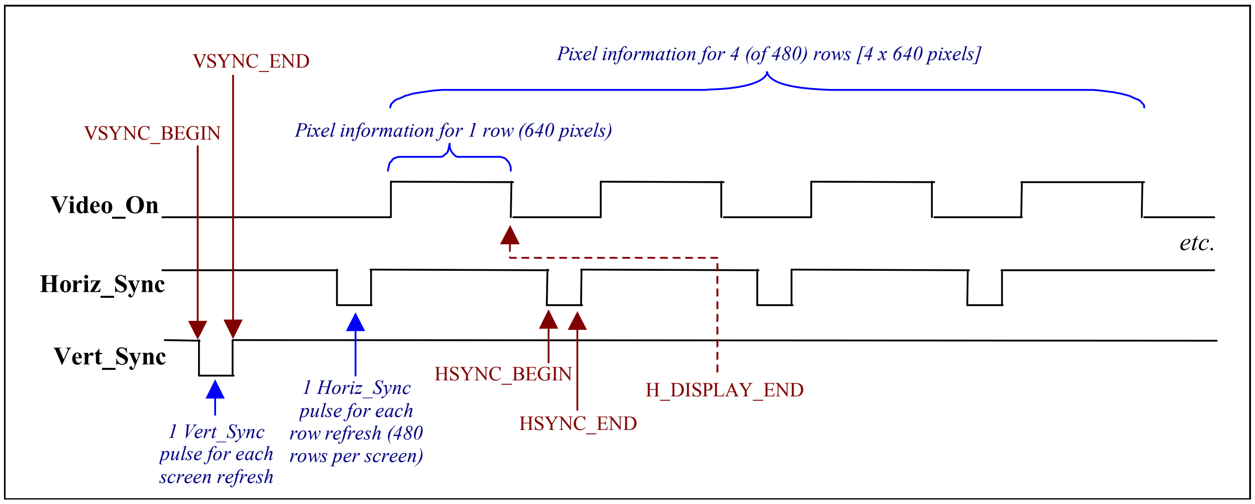

VSYNC_END
Pixel information for 4 (of 480) rows [4 x 640 pixels]
VSYNC_BEGIN
Pixel information for 1 row (640 pixels)
Video_On
etc.
Horiz_Sync
Vert_Sync
HSYNC_BEGIN
H_DISPLAY_END
HSYNC_END
1 Horiz_Sync pulse for each row refresh (480 rows per screen)
1 Vert_Sync pulse for each screen refresh

Figure 3. Timing Diagram for four rows of a VGA Display.

```
constant H_DISPLAY_END : integer := 639;
constant HSYNC_BEGIN   : integer := 659;
constant H_VERT_INC    : integer := 699;
constant HSYNC_END     : integer := 755;
constant H_MAX         : integer := 799;

constant V_DISPLAY_END : integer := 479;
constant VSYNC_BEGIN   : integer := 493;
constant VSYNC_END     : integer := 494;
constant V_MAX         : integer := 524;
```

Figure 4. Constants from VGA_LIB used for VGA Signal Generation.